UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* STEPS INTO COMPUTATIONAL GEOMETRY: NOTEBOOK II		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER R-792; UILU-ENG 77-2239
7. AUTHOR(s) F. P. Preparata, Editor		8. CONTRACT OR GRANT NUMBER(s) NSF MCS76-17321 DAAB-07-72-C-0259
9. PERFORMING ORGANIZATION NAME AND ADDRESS Coordinated Science Laboratory University of Illinois at Urbana-Champaign Urbana, Illinois 61801		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Joint Services Electronics Program		12. REPORT DATE September, 1977
		13. NUMBER OF PAGES 23
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Computational Geometry	Voronoi Diagrams
Aanalysis of Algorithms	Triangulation
Computational Complexity	Simple Polygons
Nearest-Neighbor Problems	Convex-Hull
Convex Polygons	Real-Time Algorithms

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

In this notebook we present a collection of new results in computational geometry, which all concern problems of planar geometry. The first problem is that of triangulating a simple n-vertex polygon; we show that this can be done in time O(nlogn), by first decomposing in time O(nlogn) the given polygon into a collection of special polygons, called monotone, which can be individually triangulated in time proportional to their numbers of edges. The second result concerns that all-nearest neighbor proglem for an n-vertex polygon: a surprising result is that, also no method faster than O(nlogn) is known for constructing

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

20. ABSTRACT (continued)

the Voronoi diagram of a convex polygon, the all-nearest-neighbor problem can be solved in time O(n). Finally, we show the feasibility of an optimal real-time algorithm for constructing the convex hull of a set of n points in the plane. This algorithm constructs the hull by successive updates, using total O(nlogn) time - which is optimal - with an interpoint delay O(logn), which gives the real-time property.

UILU-ENG 77-2239

STEPS INTO COMPUTATIONAL GEOMETRY: NOTEBOOK II

by

F. P. Preparata, Editor

This work was supported in part by the National Science Foundation under Grant MCS76-17321 and in part by the Joint Services Electronics Program (U.S. Army, U.S. Navy and U.S. Air Force) under Contract DAAB-07-72 C-0259.

STEPS INTO COMPUTATIONAL GEOMETRY:
NOTEBOOK II

Abstract

In this notebook we present a collection of new results in computational geometry, which all concern problems of planar geometry. The first problem is that of triangulating a simple n-vertex polygon; we show that this can be done in time O(nlogn), by first decomposing in time O(nlogn) the given polygon into a collection of special polygons, called monotone, which can be individually triangulated in time proportional to their numbers of edges. The second result concerns the all-nearest neighbor problem for an n-vertex polygon: a surprising result is that, also no method faster than O(nlogn) is known for constructing the Voronoi diagram of a convex polygon, the all-nearest-neighbor problem can be solved in time O(n). Finally, we show the feasibility of an optimal real-time algorithm for constructing the convex hull of a set of n points in the plane. This algorithm constructs the hull by successive updates, using total O(nlogn) time - which is optimal - with an interpoint delay O(logn), which gives the real-time property.

STEPS INTO COMPUTATIONAL GEOMETRY

NOTEBOOK II

F. P. Preparata, Editor

Our earlier "Notebook", by the title of "Steps into computational geometry", was issued in March of this year and consisted of an anthology of selected results into this thriving area of computational complexity.

After a few months, we have assembled a new collection of results to be enclosed in this report. As was the case with the first notebook, there is no strong unifying scheme for the results to be presented, with the exception that they all concern problems of planar geometry. One of the main reasons of this collection is simplicity of access by interested readers.

The first problem is that of triangulating a simple n-vertex polygon; we show that this can be done in time O(nlogn), by first decomposing in time O(nlogn) the given polygon into a collection of special polygons, called monotone, which can be individually triangulated in time proportional to their numbers of edges. The second result concerns the all-nearest neighbor problem for an n-vertex polygon: a surprising result is that, also no method faster than O(nlogn) is known for constructing the Voronoi diagram of a convex polygon, the all-nearest-neighbor problem can be solved in time O(n). Finally, we show the feasibility of an optimal real-time algorithm for constructing the convex hull of a set of n points in the plane. This algorithm constructs the hull by successive updates, using total O(nlogn) time - which is optimal - with an interpoint delay O(logn), which gives the real-time property.

TRIANGULATING A SIMPLE POLYGON

Franco P. Preparata

Coordinated Science Laboratory
University of Illinois at Urbana-Champaign

September 27, 1977

The efficient algorithmic construction of a triangulation of a set of points in the plane is an interesting geometric problem, which has received considerable importance from recent development in finite element methods and interpolation techniques [1,2]. In fact, if a function f of two variables x and y has been evaluated at a finite set S of points and an approximation is desired at a new point, it is convenient to visualize the diagram of f as a surface consisting of triangular plane facets. Therefore, given a triangulation of S, the function f can be evaluated by linear interpolation.

The problem of triangulating a set S of n points has been elegantly solved by Shamos [3,4]. Indeed, a triangulation is the dual graph of the Voronoi diagram of S [5], which is a well-known construct for solving proximity problems and can be algorithmically constructed with O(nlogn) steps on a random-access machine with real number arithmetic [3,4].

A more difficult problem - for which only $O(n^2)$ brute force solutions are known so far - is the triangulation of a simple polygon, which we state as follows: "Given an n-vertex simple polygon P, subdivide its interior into triangles whose vertices are also vertices of the polygon".

Notice that the general method developed for a set of points [3] is not applicable to this problem, not only because no triangulation edges may exist

---

Also, Department of Electrical Engineering and Computer Science

This work was supported in part by the National Science Foundation under Grant MCS76-17321 and in part by the Joint Services Electronics Program under Contract DAAB-07-72-C-0259.

in the exterior of P, but also because the edges of P must belong to the triangulation. In this note we show that the problem can also be solved with at most O(nlogn) steps.

The solution will be obtained in two stages, which are kept separate for expository reasons, although they are algorithmically combinable. In the first stage we decompose in time at most O(nlogn) the given simple polygon P into a collection of simple polygons $P_1, P_2, \ldots, P_k$, of a special type called monotone. Since, as we shall show, a monotone simple polygon can be triangulated in time proportional to the number of its vertices, in the second step we triangulate each of the P_i, thereby obtaining the desired triangulation of P.

For convenience, and without loss of generality, we choose the y-axis as a preferred direction. We say that a polygonal line or chain whose vertices are the sequence $(u_1, u_2, \ldots, u_p)$ is monotone (with respect to the y-axis) if $y(u_1) \geq y(u_2) \geq \ldots y(u_p)$. A simple polygon P is monotone if its boundary consists of two monotone chains with common extremes. We will now prove the following proposition.

Proposition. In n-vertex monotone polygon P can be triangulated in time O(n), and this is optimal.

Proof: The proof is algorithmic. The algorithm consists of at most $n-2+\nu$ steps, where ν is the number of nonconvex (reflex) vertices of the polygon. Each step runs in time bounded by a constant and is either a triangulation step, where a triangulation edge is created, or a data manipulation step. For easy reference, we distinguish in P a left and a right chain, with obvious meanings.

We assume inductively that at the i-th step the algorithm examines a structure consisting of two monotone chains $(u_0,u_1,u_2,\ldots,u_s)$ and (u_0,w) with $s \geq 1$, $y(u_s) \geq y(w)$ and, if $s \geq 2$, $\measuredangle (u_{j+1}u_ju_{j-1}) > \pi$ (1) for $j=1,\ldots,s-1$. Without loss of generality we shall assume that $(u_0,\ldots,u_s)$ is on the left of (u_0,w) (see figure 1). The data structure which realizes $(u_0,\ldots,u_s)$ is a stack, stored in an array A, with two pointers V and L, such that L = V+1 and $A[L] = u_s$, $A[V] = u_{s-1}$. Let ℓ be the successor of u_s in the original left chain of the polygon P. We distinguish the following cases:

(1) $y(\ell) > y(w)$ and $\measuredangle (\ell u_s u_{s-1}) > \pi$ (figure 1a). This is not a triangulation step. Vertex ℓ is added to the left chain, i.e., $V \leftarrow V+1$, $L \leftarrow L+1$, and $A[L] \leftarrow \ell$ (in other words, ℓ is "pushed" into the stack).

(2) $y(\ell) > y(w)$ and $\measuredangle (\ell u_s u_{s-1}) < \pi$ (figure 1b). This is a triangulation step. In fact, triangle $\ell u_s u_{s-1}$ cannot contain any vertex of the polygon since the chains are monotone. Thus, the triangulation edge $\overline{\ell u_{s-1}}$ can be created. At this point we check $\measuredangle(\ell u_{s-1}u_{s-2})$; if this angle is $> \pi$, then we update the stack by setting $A[L] \leftarrow \ell$; otherwise we set $V \leftarrow V-1$, and create the triangulation edge $\overline{\ell u_{s-2}}$. This process is repeated until either $\measuredangle (\ell u_i u_{i-1}) > \pi$ or $A[V] = u_0$ (the stack is empty); at this point we set $L \leftarrow V+1$, and then $A[L] \leftarrow \ell$.

(3) $y(\ell) \leq y(w)$ (figure 1c). In this case the region enclosed by the polygon $u_0 w u_s u_{s-1} \ldots u_1$ does not contain any vertex of the polygon. Moreover, by the hypothesis that $\measuredangle (u_{j+1}u_ju_{j-1}) > \pi$, for $j = 1,\ldots,s-1$, all vertices $u_1,\ldots,u_s$ are visible from w. So triangulation edges $\overline{wu_j}$, for $j=1,\ldots,s$ can be created.

---

(1) "$\measuredangle$ (a,b,c)" denotes the counterclockwise angle formed by segments $\vec{ba}$ and $\vec{bc}$

Clearly, n-3 triangulation edges are created in this process, each requiring a fixed amount of time; case (1) can occur at most as many times as there are reflex verteces in P, i.e., $\leq$ n-3 times. Thus we conclude that P can be triangulated in time O(n), and this is optimal.

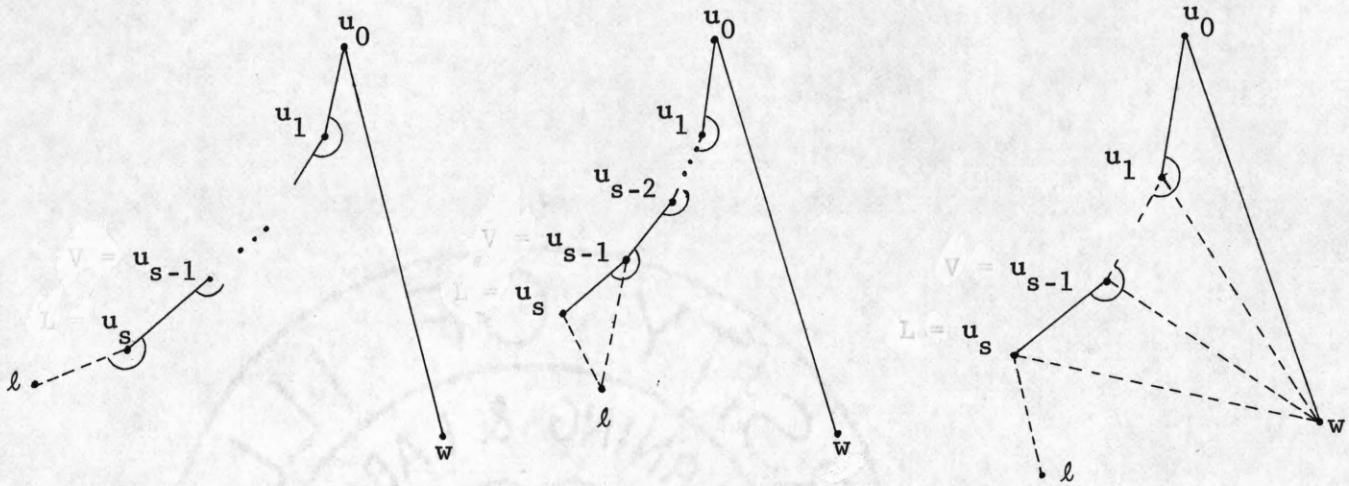

Figure 1. Triangulation of a monotone polygon.

Notice that if the i-th steps examines a two chain structure as described, by (1), (2), and (3) above so will the (i+1)-st step. □

We shall now show that any simple n-vertex polygon Q can be decomposed in time O(nlogn) into a collection of monotone polygons, by means of a rather straightforward modification of an algorithm by Lee and Preparata called "Regularization of a simple polygon" ([6], p.603). The objective of the latter procedure was the addition of edges to a given simple polygon to ensure that each vertex was contained in some monotone chain between the two vertices of largest and smallest y-coordinate (extreme vertices). This was done as follows. Assume, for simplicity of explanation, that no two vertices in the vertex sequence $(v_0, v_1, \ldots, v_{n-1})$ of the polygon Q have identical y-coordinates (these ties, however, can be taken care of very easily). Let a <u>cusp</u> be a vertex v_j for which either $y(v_{j-1}) < y(v_j)$ and $y(v_{j+1}) < y(v_j)$ or $y(v_{j-1}) > y(v_j)$ and $y(v_{j+1}) > y(v_j)$. The above objective is met if, for each vertex v of Q -

except the two extremes vertices - there are two vertices v' and v" such that $y(v') > y(v) > y(v'')$. This condition is satisfied by all vertices except for cusps: thus only a cusp - if it is not one of the two extreme vertex - is to be connected to another vertex of Q by means of an edge not crossing any other edge. This task can be carried out by the above mentioned "regularization procedure" in time O(logn) per vertex. The variant, in our present problem is that not all nonextreme cusps need be connected, but only those whose reflex angle is inside Q. Thus all that is needed is a preliminary scan of the vertex sequence of Q to tag only the cusp which need additional connection; next we can apply a regularization procedure which will omit connecting untagged vertices. The different results of the original [6] and of the modified procedures are shown in Figures 2a and 2b, respectively. The preliminary scan runs in time O(n), whereas the decomposition step runs in time O(nlogn). The latter task is responsible for the order of complexity of the entire process. In figure 2(c) we show the end product of the triangulation process.

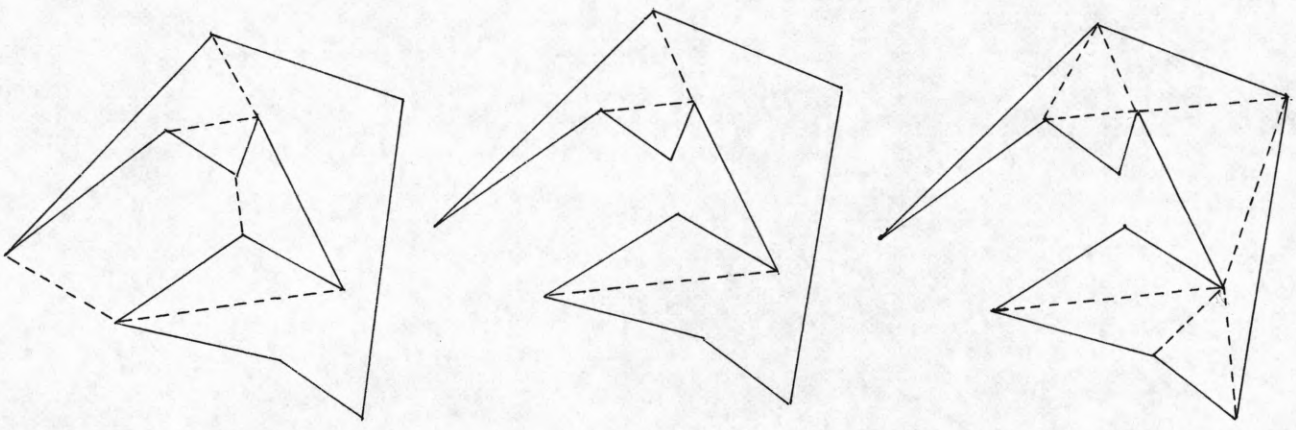

Figure 2. Simple polygon Q: (a) regularized; (b) decomposed into monotone polygons; (c) triangulated.

References

1. G. Strang and G. Fix, An Analysis of the Finite Element Method, Prentice-Hall (1973).

2. D. H. McLain, "Two-dimensional interpolation from random data," Computer Journal, Vol. 19, N. 2, pp. 178-181, 1976.

3. M. I. Shamos, "Geometric Complexity," Proc. Seventh Annual ACM Symposium on Theory of Computing, pp. 224-233, May 1975.

4. M. I. Shamos, Computational Geometry, Dept. of Comp. Sci., Yale University 1977. To be published by Springer-Verlag.

5. G. Voronoi, "Nouvelles applications des parametres continus a la theorie des formes quadratiques. Deuxieme Memoire: Recherches sur les paralleloeolres primitifs.", J. reine und angew. Math., 134, pp. 198-287. (1908).

6. D. T. Lee and F. P. Preparata, "Location of a point in a planar subdivision and its applications," SIAM Journal on Computing, Vol. 6, N. 3, pp. 594-606, September 1977.

THE ALL NEAREST-NEIGHBOR PROBLEM FOR CONVEX POLYGONS*

D. T. Lee

August 1977

Introduction

The problem of finding the nearest neighbor for each of N arbitrary points in the Euclidean plane has been shown [1] to require time O(NlogN), and the algorithms achieving the lower bound have also been given in [1,2]. However, the lower bound does not apply to the same problem when the given points, rather than being arbitrarily placed, form a convex polygon. In this paper, we shall show that this additional information indeed enables us to obtain a linear time algorithm, whose running time is obviously optimal within a multiplicative constant.

Main result

Let a convex polygon P be denoted by a sequence of vertices $(P_0,P_1,\ldots,P_{N-1})$ in which $\overline{P_iP_{i+1}}$,[1] $0 \leq i < N$, is an edge. Define an index set $I=\{0,1,\ldots,N-1\}$. Let $d(P_i,P_j)$, $i,j \in I$, denote the distance between P_i and P_j and D(P) denote the diameter of P, i.e., $D(P) = \max_{i,j \in I} d(P_i,P_j)$, the largest distance between

*This work was supported in part by the National Science Foundation under Grant MCS-76-17321 and in part by the Joint Services Electronics Program under Contract DAAB-07-72-C-0259.

[1] All indices in the text are taken modulo N.

the vertices of P. The nearest neighbor $NN(p_i)$ of p_i is p_j such that $d(p_i,p_j) = \min_{k \in I-\{i\}} d(p_i,p_k)$. Consider now the following conditions:

Condition (i): the two farthest points of P are the extremes of an edge, i.e., $D(P) = d(p_i,p_{i+1})$ for some i.

Condition (ii): all vertices of P lie inside a circle with diameter D(P).

A convex polygon P, which satisfies both (i) and (ii), is said to have the semi-circle property. Figure 1 shows a convex polygon having the semi-circle property.

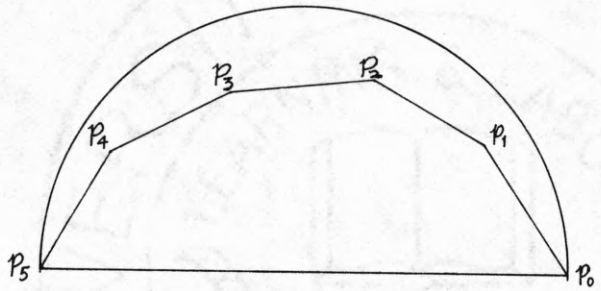

Figure 1. A convex polygon with the semi-circle property.

Lemma 1 [3] Given a convex polygon $P = (p_0,p_1,\ldots,p_{N-1})$, there exists a linear time algorithm to decompose it into at most four convex polygons which have the semi-circle porperty.

Proof: First of all, we apply the linear time algorithm [4] to find the diameter. Let $D(P) = d(p_u,p_v)$. The chord $\overline{p_up_v}$ will, in general, divide P into two convex polygons $P_1 = (p_u,p_{u+1},\ldots,p_v)$ and $P_2=(p_v,p_{v+1},\ldots,p_{N-1},p_0,\ldots,p_u)$ (Figure 2), where $D(P_1) = D(P_2) = d(p_u,p_v)$. Let $p_\ell \in P_1$ be the vertex with

largest distance from the chord $\overline{p_u p_v}$. Let $p_m \in P_2$ be defined similarly. It is obvious that p_ℓ and p_m can be found in O(N) time. p_ℓ will divide P_1 into two convex polygons $P_{11} = (p_u, p_{u+1}, \ldots, p_\ell)$ and $P_{12} = (p_\ell, p_{\ell+1}, \ldots, p_v)$. Similarly, p_m divides P_2 into $P_{21} = (p_v, p_{v+1}, \ldots, p_m)$ and $P_{22} = (p_m, p_{m+1}, \ldots, p_{N-1}, p_0, \ldots, p_u)$. We claim that P_{11}, P_{12}, P_{21}, and P_{22} satisfy the semi-circle property. Without loss of generality we shall just consider P_{12}.

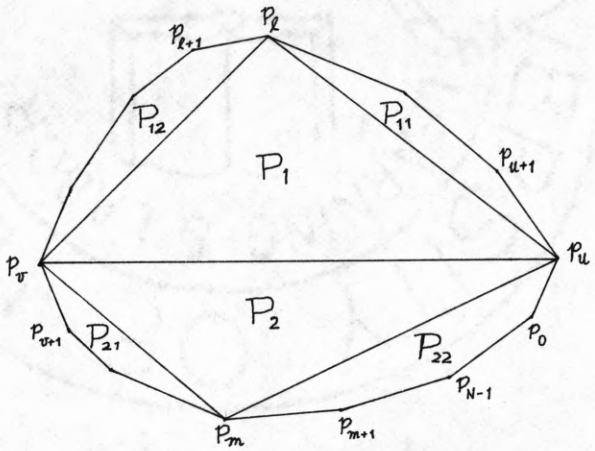

Figure 2. Decomposition of a convex polygon into four convex polygons satisfying semi-circle property.

In Figure 3, since $\overline{p_u p_v}$ is the longest chord, all vertices $p_u, p_{u+1}, \ldots, p_v$ must lie within the region $p_u Q p_v$ where Q is the intersection of the two circles with radius $d(p_u, p_v)$ and centered at p_u and p_v respectively. By convexity the vertices of P_{12} must lie in the region $p_\ell Q' p_v$, and $D(P_{12}) = d(p_\ell, p_v)$. The line $\overline{A' p_v}$ perpendicular to $\overline{p_u p_v}$ intersects $\overline{p_\ell Q'}$ at A'. The region $p_\ell Q' p_v$ is contained in the right triangle $p_v A' p_\ell$, which is obviously contained in the semi-circle with diameter $\overline{p_\ell p_v}$ and center A. □

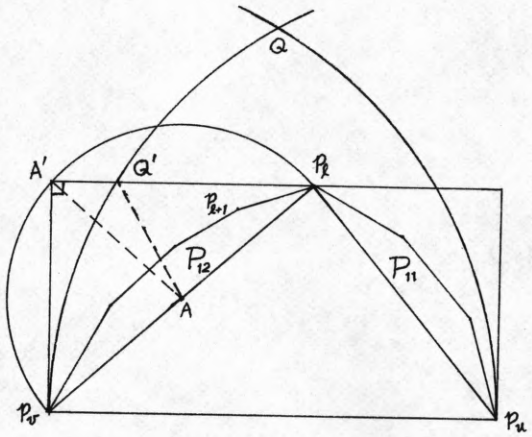

Figure 3. Proof that P_{12} has semi-circle property.

Lemma 2 [3]: Given a convex polygon $P = (p_0, p_1, \ldots, p_{N-1})$ with the semi-circle property, for any vertex p_i, its nearest neighbor p_j is adjacent to p_i, i.e., either $j = i+1$ or $j = i-1$.

Proof: Without loss of generality, we may assume that $D(P) = d(p_0, p_{N-1})$. Suppose for some p_i, $NN(p_i) = p_k$ where $k > i+1$ (Figure 4). Consider the triangle $p_i p_{i+1} p_k$. Since $d(p_i, p_k) < d(p_i, p_{i+1})$ by assumption, the angle $\measuredangle\, p_i p_{i+1} p_k$ is less than the angle $\measuredangle\, p_i p_k p_{i+1}$. By convexity, p_i and p_k must lie above chords $\overline{p_0 p_{i+1}}$ and $\overline{p_{i+1} p_{N-1}}$ respectively. Thus $\measuredangle\, p_i p_{i+1} p_k$ must be greater than $\measuredangle\, p_0 p_{i+1} p_{N-1}$ which is greater than $\pi/2$ by the semi-circle property of the given polygon. That is, $\measuredangle\, p_i p_k p_{i+1} > \measuredangle\, p_i p_{i+1} p_k > \pi/2$ which is impossible. Therefore $NN(p_i)$ must be adjacent to p_i, for all $i \in I$. □

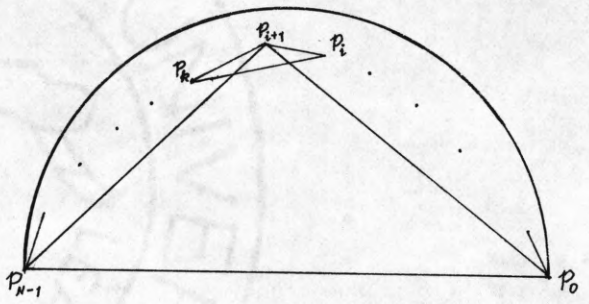

Figure 4. Illustration of the proof of lemma 2.

Lemma 3. Given a convex polygon $P = (p_0, p_1, \ldots, p_{N-1})$ satisfying condition (i) above, the nearest neighbor $NN(p_i)$ of p_i, for all $i \in I$, can be found in $O(N)$ time.

Proof: Suppose $D(P) = d(p_0, p_{N-1})$. Let p_i be the vertex with the largest distance to the chord $\overline{p_0 p_{N-1}}$. By lemma 1, the two convex polygons $P_1 = (p_0, p_1, \ldots, p_i)$ and $P_2 = (p_i, p_{i+1}, \ldots, p_{N-1})$ satisfy the semi-circle property. The nearest neighbor of each vertex in P_s, $(s = 1,2)$ can be found separately by a simple scan through the vertices of P_s by lemma 2. This step takes $O(N)$ time. We must still check whether the nearest neighbor of a vertex in, say P_1, belongs to P_2, and this can be done as follows. Let p_{N-1} be the origin, and the chord $\overline{p_{N-1} p_0}$, directed from p_{N-1} to p_0, define the positive x-axis. By assumption, p_i has the largest y-coordinate. We project all the vertices on the vertical line $p_i q$ through p_i (Figure 5). The projections of the vertices in P_1 and P_2, denoted by $\ell(P_1) = \{\ell(p_0),$

$\ell(p_1), \ldots, \ell(p_i)\}$ and $\ell(P_2) = \{\ell(p_i), \ell(p_{i+1}), \ldots, \ell(p_{N-1})\}$ are ordered as $(\ell(p_i), \ell(p_{i-1}), \ldots, \ell(p_0))$ and $(\ell(p_i), \ell(p_{i+1}), \ldots, \ell(p_{N-1}))$, respectively, from top to bottom. Let

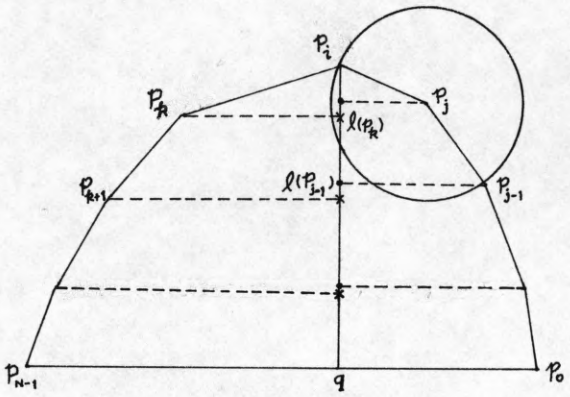

Figure 5. Projections of vertices onto $p_i q$ are ordered in y-coordinate.

$\delta(p_j)$ be the distance from p_j to its nearest neighbor $NN(p_j)$ where $NN(p_j)$ and p_j are in the same polygon P_s, $s = 1,2$, and let $\underline{\delta\text{-circle}}\ (p_j)$ denote the circle with radius $\delta(p_j)$ and centered at p_j. Consider the case when $p_j \in P_1$. The only possible nearest neighbor of p_j is among those $p_k \in P_2$ whose projections $\ell(p_k)$ are contained in the δ-circle (p_j) (Figure 5). For each $\ell(p_k)$, there can be at most four δ-circles that pass through it [2]. Therefore, in order to find $NN(p_j)$ for all $p_j \in P_1$ we need only examine at most four times the number of points in P_2. Specifically, let the vertices of P_1 and P_2 be kept as two ordered lists $(p_i, p_{i-1}, \ldots, p_0)$ and $(p_i, p_{i+1}, \ldots, p_{N-1})$, respectively and let s_1 and s_2 be pointers associated with these two lists, respectively. In a first scan we shall determine the nearest neighbor of each $p_j \in P_1$; anologously, in a second scan (not to be described) we shall process P_2. In each step, we shall

examine the δ-circle (p_{s_1}) for $p_{s_1} \in P_1$ and the projection $\ell(p_{s_2})$ for $p_{s_2} \in P_2$. We start by setting $s_1 \leftarrow i$ and $s_2 \leftarrow i+1$. If the δ-circle (p_{s_1}) does not intersect $\overline{p_i q}$ (Figure 5), the nearest neighbor of p_{s_1} belongs to P_1 and was found earlier. So we update $s_1 \leftarrow s_1 - 1$. Suppose, for some p_{s_1}, the δ-circle (p_{s_1}) intersects $\overline{p_i q}$ at u and v where u is above v (Figure 6). Then we examine the projection $\ell(p_{s_2})$ to see if it belongs to the segment $\overline{uv}$. If $\ell(p_{s_2})$ is above u, we update $s_2 \leftarrow s_2 + 1$, since p_{s_2} can not be the nearest neighbor of p_{s_1}. Let f be the smallest value of s_2 so that $\ell(p_f)$ belongs $\overline{uv}$. Since p_f is a possible candidate, we check if $d(p_{s_1}, p_f)$ is less than $d(p_{s_1}, NN(p_{s_1}))$. If so, we update $NN(p_{s_1})$

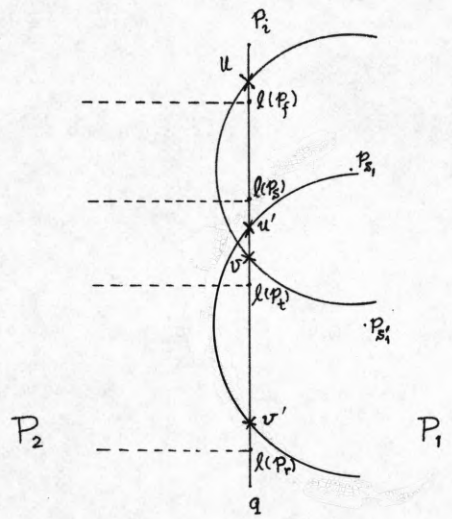

Figure 6. Illustration of the procedure in lemma 3.

accordingly. We keep advancing s_2 and do the same checking and updating until we reach a vertex p_t whose projection $\ell(p_t)$ is below v. At this point, the nearest neighbor $NN(p_{s_1})$ of p_{s_1} is determined and we update

$s_1 \leftarrow s_1 - 1$. Suppose δ-circle $(p_{s_1'})$, for some $p_{s_1'} \in P_1$, is the next circle that intersects $\overline{p_i q}$ at u' and v' where u' is above v' (Figure 6). If $\ell(p_t)$ is above u', we advance s_2. Otherwise, we scan backward and if $\ell(p_t)$ is on $\overline{u'v'}$ we do the checking and updating as before until we reach a vertex p_s whose projection $\ell(p_s)$ is above u'; then we scan forward from $\ell(p_t)$ on and do the same until we meet a vertex p_r whose projection $\ell(p_r)$ is below v'. At this point, the nearest neighbor of $p_{s_1'}$ is determined, we update s_1. Note that now $s_2 = r$. The process is repeated until all the nearest neighbor of p_j's in P_1 are determined. As pointed out above, each projection can only be examined at most four times, so the time required for the first pass is $O(n_2)$ where n_2 is the number of vertices in P_2. Similarly, the time for the second pass in $O(n_1)$ where n_1 is the number of vertices in P_1. Therefore, this step takes $O(n_1 + n_2) = O(N)$ time.

Since each step takes O(N) time, the proof of this lemma is completed. □

Based on the above lemmas, we have the following theorem.

Theorem: Given a convex polygon $P = (p_0, p_1, \cdots, p_{N-1})$, the nearest neighbor of each vertex can be found in O(N) time.

Proof: Let $D(P) = d(p_u, p_v)$. The chord $\overline{p_u p_v}$ divides the polygon P into two polygons $P_1 = (p_u, p_{u+1}, \cdots, p_v)$ and $P_2 = (p_v, p_{v+1}, \cdots, p_{N-1}, p_0, \cdots, p_u)$. By lemma 3, the nearest neighbor $NN(p_j) \in P_s$ of $p_j \in P_s$, $(s = 1,2)$ can be found in O(N) time. Now, we project all the vertices in $P_s, (s = 1,2)$ onto the chord $\overline{p_u p_v}$. Since the projections of the vertices in $P_s, (s = 1,2)$ are ordered respectively, by a technique similar to that described in lemma 3, we can find for each vertex $p_i \in P$, its nearest neighbor in O(N) time.

Since the diameter of P can be found in O(N) time, the total running time is O(N). □

Conclusion

It is rather interesting that the nearest-neighbor problem for a set of N arbitrary points is lower-bounded by O(N log N), whereas the problem can be solved in linear time if the given set of points forms a convex polygon. In [1], the nearest neighbor problem was solved by the Voronoi diagram technique. The construction of the Voronoi diagram for a set of N points has also been shown to require O(N log N) time [1]. But whether the construction of the Voronoi diagram for the set of vertices of a convex polygon can be solved in less than O(N log N) time still remains an open problem. However, we know at least that the nearest neighbor problem for the set of vertices of a convex polygon is not as time-consuming as the presently known techniques for constructing the Voronoi diagram for it.

Acknowledgement

The author wishes to express his deep gratitude to F. P. Preparata for his invaluable suggestions.

References

[1] Shamos, M. I. and Hoey, D. J., "Closest-Point Problems," Proc. 16th Annual IEEE Symposium on Foundations of Computer Science, Oct. 1975, pp. 151-162.

[2] Bentley, J. L. and Shamos, M. I., "Divide-and-Conquer in Multi-dimensional Space," Proc. 8th Annual ACM Symposium on Theory of Computing, May 1976, pp. 220-230.

[3] Preparata, F. P., private communication.

[4] Shamos, M. I., "Geometric complexity," Proc. 7th Annual ACM Symposium on Theory of Computing, May 1975, pp. 224-233

[5] Shamos, M. I., Problems in Computational Geometry, Dept. of Computer Science, Yale University, New Haven, Conn., May 1975 (to be published by Springer-Verlag).

AN OPTIMAL REAL-TIME ALGORITHM FOR PLANAR CONVEX HULLS

F. P. Preparata

October 3, 1977

1. Introduction

Algorithms for finding the convex hull of a finite set of n points in the plane have been developed by several authors in recent years [1,2,3,4]. Most of these algorithms are also optimal, that is, as pointed out in [5], they have worst-case running time $O(n \log n)$, which is also the best achievable performance.

A common feature of the above mentioned algorithms is that they are all off-line, i.e., they operate on the data collectively. In other words, information about all points of the set must be available before any of those algorithms can be applied.

Instead, it is desirable to develop an algorithm which receives one point at a time and updates the convex hull accordingly, so that, after points $p_1, p_2, \ldots, p_i$ have been received their convex hull is available. Such an algorithm is appropriately called on-line. A general feature of on-line algorithms is that no bound is placed on the update time, or, equivalently, a new item (point) is input on request as soon as the update relative to the last item has been completed. We shall refer to the time

---

This work was supported in part by the National Science Foundation under Grant MCS76-17321 and in part by the Joint Services Electronics Program under Contract DAAB-07-72-C-0259.

interval between two consecutive inputs as the interarrival delay.

Frequently, known on-line algorithms are less efficient on the entire set than the corresponding off-line algorithms (some price must generally be paid to acquire the on-line property). For the planar convex hull problem, however, Shamos has designed an elegant on-line algorithm [6], which runs in time $O(n \log n)$, thereby matching the performance of off-line algorithms for the same problem.

A more demanding case of on-line applications occurs when the interarrival delay is outside the control of the algorithm. In this case the update must be completed in time no greater than the minimum interarrival delay. Algorithms for such applications are appropriately called in real-time. Shamos points out in [6] that, since any convex hull algorithm on n points requires $\Omega(n \log n)$ operations, any real-time algorithm for this problem must be allowed $O(\log n)$ processing time between successive inputs.

Unfortunately, the algorithm described by Shamos exceeds this allowance, since its interarrival delay can be $O((\log n)^2)$. The algorithm works as follows. When the point p_i is supplied, assume inductively that the algorithm has available the convex hull H_{i-1} of the set of points $\{p_i, \ldots, p_{i-1}\}$, a point 0 internal to H_{i-1}, and the polar angles of the vertices of H_{i-1} - a convex polygon - about 0. The vertices of H_{i-1} are arranged in a height-balanced tree (e.g., an AVL tree), in the order of their polar angles. Thus point p_i can be located between two consecutive vertices of H_{i-1} in time at most O(logi) and tested for inclusion in H_{i-1}. If it is internal, it is discarded; otherwise, two vertices ℓ and r of H_{i-1} have to be located so that the segments $p_i\ell$ and $p_i r$ belong to lines of support of H_{i-1}. The points ℓ and r can each be located by performing a

standard binary search on the vertex cycle of H_{i-1}; on the other hand, each probe of this search is itself a search in the AVL tree, thereby resulting in a worst-case running time $O((\log i)^2) = O((\log n)^2)$ for an update.

The intuitive reason why the algorithm sketched above fails to achieve an $O(\log n)$ update time is that a binary search is artificially forced on a search tree, rather than letting the latter be the guide of the search operation. This natural observation is the basis of the following convex hull algorithm, which runs in time $O(n \log n)$, and is therefore optimal, and has update time $O(\log n)$, and is therefore in real-time.

2. The Real-time Algorithm

Let P be a polygon in the plane and let $(v_0, \ldots, v_{s-1})$ be the counterclockwise cycle of its vertices (indices are modulo s). The vertices of P will be stored in a data structure T(P) which is a height balanced tree modified in a trivial way. Specifically in the node associated with vertex v_i we also store a pointer NEXT $[v_i]$, which gives the address of the node of v_{i+1}. Let also min T(P) denote the first member of the vertex cycle. The convex-hull algorithm will make use of two procedures: TEST and RESTRUCTURE. Procedure TEST (P,m,p) accepts as its inputs a point p, a convex polygon P, represented by the tree T(P), and the minimum element m = min T(P), this algorithm tests whether p is internal or external to P, and, in the latter case, it determines two vertices ℓ and r, previously defined (see Figure 1): notice that ℓ and r are named so that $\measuredangle(rp\ell) < \pi$[1]. If p is internal, the algorithm terminates without altering T(P); otherwise the string of vertices comprised between ℓ and r is deleted, and the vertex p is then inserted between ℓ and r. This operation is performed by procedure RESTRUCTURE (P,p,ℓ,r)

---

[1] $\measuredangle(abc)$ denotes the counterclockwise angle formed by segments $\vec{ba}$ and $\vec{bc}$.

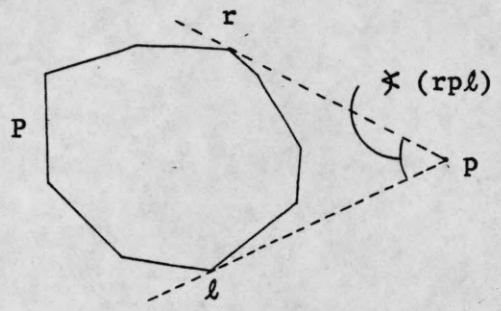

Figure 1. Definition of vertices ℓ and r.

Less informally, we have (Λ is the empty symbol):

CONVEX-HULL UPDATE

Input: $T(H_{i-1})$, p_i

Output: $T(H_i)$

1. begin $m \leftarrow \min T(H_{i-1})$
2. $(\ell,r) \leftarrow$ TEST (H_{i-1},m,p_i)
3. If $(\ell,r) \neq (\Lambda,\Lambda)$ then $H_i \leftarrow$ RESTRUCTURE (H_{i-1},p_i,ℓ,r) else $H_i \leftarrow H_{i-1}$

end

Obviously, Step 1 runs in time at most O(logi) (search in height balanced tree with at most i elements). We shall now show that both TEST and RESTRUCTURE run in time at most O(logi). We begin by considering TEST. Let $T = T(H_{i-1})$, $m = \min(T)$, and $M = \text{ROOT}(T)$. Given point p_i and a vertex v of H_{i-1}, we shall say that v is convex (with respect to p_i) if the segment p_iv intersects the interior of H_{i-1}; otherwise, if the two vertices adjacent to v lie on the same side of the line containing p_iv, v is supporting; in the remaining case, v is reflex (see figure 2). We also denote as α the angle $\measuredangle (mp_iM)$: obviously

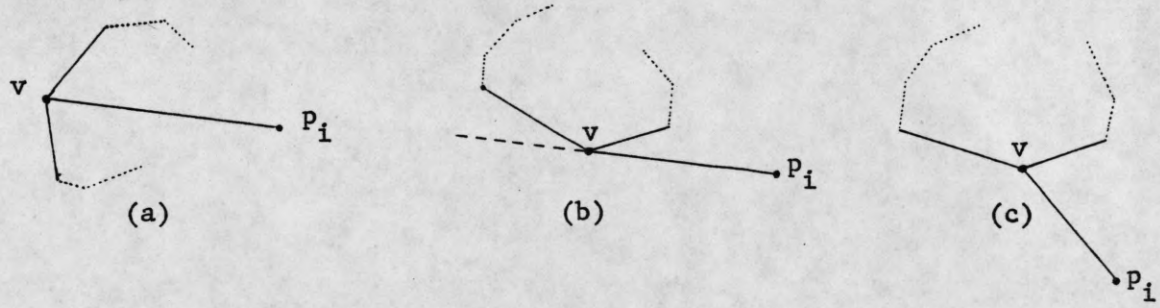

Figure 2. Illustration of convex, supporting, and reflex vertices.

α is classifiable as convex ($\leq \pi$) or reflex ($> \pi$). Depending upon the classifications of m, M, and α, we have in total 18 possible cases. These elementary cases can be conveniently grouped into eight new cases, illustrated in figure 3, each requiring distinct algorithmic actions. Notationally, let the two supporting vertices ℓ and r be on the left and on the right of an observer placed in p_i and facing the polygon H_{i-1}. We shall make use of two procedures, called LEFTSEARCH and RIGHTSEARCH, which determine ℓ and r, respectively. The arguments of these procedures are binary search trees. We also let L(M) and R(M) denote the left and the right subtree of the node M, respectively. The analysis of the eight cases and their corresponding actions is straightforward. It is worth pointing out that when p_i is internal to H_{i-1}, cases 1 or 7 will occur repeatedly, the algorithm will examine a nested family of subtrees, and will terminate when the subtree consists of only one leaf, i.e., when m = M (see Step 2 below). We can now explicitly give the algorithm:

TEST (H,m,p)

Input: H, a polygon, represented as a modified AVL tree T(H)
p a point, m the minimum element in T(H).

Output: either a pair (ℓ,r) of integers or (Λ,Λ)

```
1.  begin M ← ROOT (T(H)), T ← T(H)
2.      If m = M then r ← ℓ ← Λ (Comment: p_i is internal)
3.      else begin If α ≤ π then
4.                   If m is convex then
5.                     If M is convex then T ← R(M),  m ← NEXT(M), u ← 0 (case 1)
6.                     else T_1 ← T-R(M), T_2 ← R(M), u ← 1 (case 2)
7.                   else If M is reflex then T ← L(M), u ← 0 (case 3)
8.                        else T_1 ← T - L(M), T_2 ← L(M), u ← 1 (case 4)
9.                 else If m is reflex then
10.                      If M is reflex then T ← R(M),m ← NEXT(M) u ← 0 (case 5)
```

11. else $T_1 \leftarrow R(M), T_2 \leftarrow T - R(M)$, $u \leftarrow 1$ (case 6)

12. else If M is convex then $T \leftarrow L(M)$, $u \leftarrow 0$ (case 7)

13. else $T_1 \leftarrow L(M)$, $T_2 \leftarrow T - L(M)$, $u \leftarrow 1$ (case 8)

14. If $u = 0$ then $(\ell, r) \leftarrow$ TEST(T, m, p_i)

15. else $\ell \leftarrow$ LEFTSEARCH(T_1), $r \leftarrow$ RIGHTSEARCH(T_2)

end

16. return (ℓ, r)

end

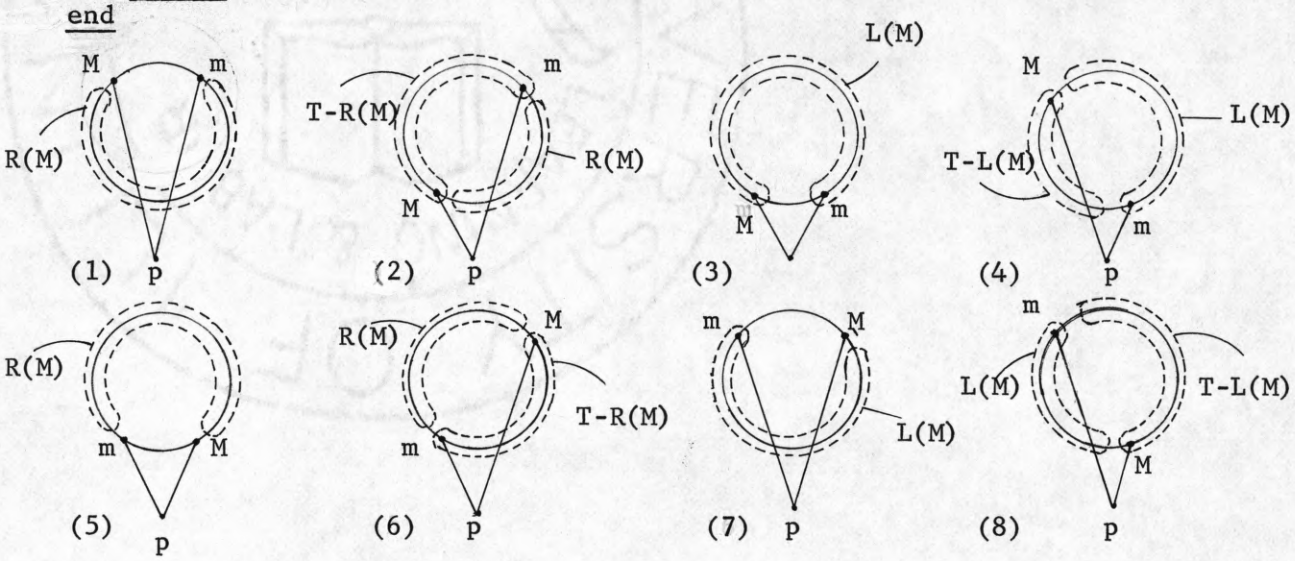

Figure 3. The eight possible cases handled by algorithm TEST.

We shall now describe the procedure LEFTSEARCH used by the algorithm TEST (RIGHTSEARCH is analogous with obvious modifications).

Procedure LEFTSEARCH

Input: a tree T, describing a sequence of vertices

Output: a vertex ℓ

1. begin $c \leftarrow$ ROOT(T)

2. If pc is supporting then $\ell \leftarrow c$

3. else begin If c is reflex then $T \leftarrow L(c)$ else $T \leftarrow R(c)$

4. $\ell \leftarrow$ LEFTSEARCH(T)

end

5. return ℓ

end

It is obvious that LEFTSEARCH involves tracing a path of the tree T, spending a bounded time at each node. Since T is a balanced tree with at most (i-1) nodes, the running time is O(logi). Referring now to the time performance of TEST, we notice that the bulk of the work is done either in Step 14 (recursive call of TEST) or in Step 15 (calls of LEFTSEARCH AND RIGHTSEARCH), whereas the decisions leading to either of these steps (Steps 3-13) take time bounded by a constant. Typically, algorithm TEST could be viewed as tracing a path from the root to some node c of $T(H_{i-1})$, recursively calling itself. If p_i is internal to H_{i-1}, then c is a leaf of $T(H_{i-1})$; otherwise, starting at node c, two paths of $T(H_{i-1})$ are traced by LEFTSEARCH and RIGHTSEARCH, respectively, until ℓ and r are found. Since the amount of work expended at each node is bounded by a constant, TEST runs in time O(logi).

Finally, we consider the procedure RESTRUCTURE, which is invoked only when p_i is external to H_{i-1}. Let n_{i-1} be the number of vertices H_{i-1}. As mentioned earlier, the vertices comprised between ℓ and r must be deleted and p_i must be inserted. With regard to the deletion, slightly different actions will be taken depending upon whether ℓ precedes r in $T(H_{i-1})$ or not. In the first case we have to split twice and splice once AVL trees with at most i-1 elements; in the second case, only two splittings occur. But split and splice of AVL trees are standard operations, known as Crane's algorithms ([7], p.465), which can be performed in time O(logi) and will not be further discussed. Similarly,insertion of p_i can be done in time O(logi), whereas the update of the function NEXT only involves two pointers, associated with ℓ and p_i respectively.

Therefore, we conclude that CONVEX-HULL UPDATE can be executed in time O(logi) after i points have been processed and can be used for an optimal real-time algorithm.

References

1. R. L. Graham, "An efficient algorithm for determining the convex hull of a finite planar set," Information Processing Letters, Vol. 1, pp. 132-133 (1972).

2. R. A. Jarvis, "On the identification of the convex hull of a finite set of points in the plane," Information Processing Letters, Vol. 2, pp. 18-21 (1973).

3. M. I. Shamos, "Problems in computational geometry," Department of Computer Science, Yale University, New Haven, Conn., May 1975, (To be published by Springer Verlag).

4. F. P. Preparata and S. J. Hong, "Convex hulls of finite sets in two and three dimensions," Communications of the ACM, Vol. 20, N. 2, pp. 87-93, February 1977.

5. M. I. Shamos, "Geometric Complexity," Proc. Seventh Annual ACM Symposium on Theory of Computing, pp. 224-233, May 1975.

6. M. I. Shamos, Computational Geometry, Dept. of Computer Sci., Yale University, 1977. To be published by Springer Verlag.

7. D. E. Knuth, The Art of Computer Programming, Volume 3: Sorting and Searching, Addison-Wesley, Reading Mass., 1973.